UNIVERSITY OF CALIFORNIA
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

CS61B **P. N. Hilfinger**
Spring 2020

Test #1 (revised)

READ THIS PAGE FIRST. *Please do not discuss this exam with people who haven't taken it.* Your exam should contain 6 problems on 13 pages. Officially, it is worth 17 points (out of a total of 200).

This is an open-book test. You have 110 minutes to complete it. You may consult any books, notes, or other non-responsive objects available to you. You may use any program text supplied in lectures, problem sets, or solutions. Please write your answers in the spaces provided in the test. Make sure to put your name, login, SID, and TA in the space provided below. Put your login and initials *clearly* on each page of this test and on any additional sheets of paper you use for your answers.

Be warned: my tests are known to cause panic. Fortunately, this reputation is entirely unjustified. Just read all the questions carefully to begin with, and first try to answer those parts about which you feel most confident. Do not be alarmed if some of the answers are obvious. Should you feel an attack of anxiety coming on, feel free to jump up and run around the outside of the building once or twice.

Your name: _____ Login: cs61b-_____

Your SID: _____

Login of person to your Left: _____ Right: _____

Discussion TA: _____

Reference Material.

```
/* arraycopy(FROM_ARRAY, FROM_INDEX, TO_ARRAY, TO_INDEX, LENGTH) */
import static java.lang.System.arraycopy;

public class IntList {
    /** First element of list. */
    public final int head;
    /** Remaining elements of list. */
    public IntList tail;

    /** A List with head HEAD0 and tail TAIL0. */
    public IntList(int head0, IntList tail0)
    { head = head0; tail = tail0; }

    /** A List with null tail, and head = HEAD0. */
    public IntList(int head0) { this(head0, null); }

    /** Returns a new IntList containing the ints in ARGS. */
    public static IntList list(Integer ... args) {
        // Implementation not shown
    }

    /** Returns true iff L (together with the items reachable from it) is an
     *  IntList with the same items as this list in the same order. */
    @Override
    public boolean equals(Object L) {
        // Implementation not shown
    }

    /** Return the length of the non-circular list headed by this. */
    public int size() {
        // Implementation not shown
    }

}
```

1. [4 points] Given two one-dimensional arrays `LL` and `UR`, fill in the program on the next page to insert the elements of `LL` into the lower-left triangle of a square two-dimensional array `S` and `UR` into the upper-right triangle of `S`, without modifying elements along the main diagonal of `S`. You can assume `LL` and `UR` both contain at least enough elements to fill their respective triangles.

For example, consider

```
int[] LL = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 0 };
int[] UR = { 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };
int[][] S = {
    { 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0},
    { 0, 0, 0, 0, 0}
};
```

After calling `fillGrid(LL, UR, S)`, `S` should contain

```
{
  { 0, 11, 12, 13, 14 },
  { 1,  0, 15, 16, 17 },
  { 2,  3,  0, 18, 19 },
  { 4,  5,  6,  0, 20 },
  { 7,  8,  9, 10,  0 }
}
```

(The last two elements of `LL` are excess and therefore ignored.)

```
/** Fill the lower-left triangle of S with elements of LL and the
 *  upper-right triangle of S with elements of UR (from left-to
 *  right, top-to-bottom in each case).  Assumes that S is square and
 *  LL and UR have at least sufficient elements. */
public static void fillGrid(int[] LL, int[] UR, int[][] S) {
    int N = S.length;
    int kL, kR;
    kL = kR = 0;

    for (int i = 0; i < N; i += 1) {

          int c = 0;
        ----------------------------------------------

          while(i!= c){
        ----------------------------------------------

          S[i][c] = LL[kL];
        ----------------------------------------------

          kL++;}
        ----------------------------------------------

          S[i][c] = 0;
        ----------------------------------------------

          while(c<)
        ----------------------------------------------


        ----------------------------------------------


        ----------------------------------------------


        ----------------------------------------------
    }
}
```

2. [1 point] Which one of the following quotations attributed to Gloria Steinem did not, in fact, originate with her?

- The truth will set you free, but first it will piss you off.
- A woman without a man is like a fish without a bicycle.
- If the shoe doesn't fit, must we change the foot?
- Whatever you want to do, just do it.... Making a damn fool of yourself is absolutely essential.
- Women have two choices: Either she's a feminist or a masochist.
- The first problem for all of us, men and women, is not to learn, but to unlearn.

3. [5 points] Consider the following function, which is intended to carry out one step in a (not terribly efficient) sorting procedure:

```
static void swapUp(int[] A) {
    for (int i = 0; i < A.length - 1; i += 1) {
        if (A[i] > A[i + 1]) {
            int tmp = A[i]; A[i] = A[i + 1]; A[i + 1] = tmp;
        }
    }
}
```

Unless `A` is already in non-descending order, this moves at least one larger element in `A` closer to where it ought to be. For example, if `B` initially contains the sequence

```
10, 3, 2, 11, 1, 6, 0, 20, 2
```

then after the call `swapUp(B)`, it will contain the sequence

```
3, 2, 10, 1, 6, 0, 11, 2, 20
```

Suppose that instead of performing this procedure on an array, we instead do it on `IntList`s, as defined on page 2.

a. Fill in the method `swapUpList` so that it ***non-destructively*** returns the list resulting from applying the same algorithm as `swapUp`, but with the input sequence being an `IntList` rather than an array.

b. Fill in the method `dswapUpList` so that it ***destructively*** returns the list resulting from applying the same algorithm as `swapUp`. This version should never execute a **new** operator, except for creating one temporary `IntList` that is used to make sure there is a predecessor to every item in the list, and is not included in the final result. Otherwise, your program should re-use the same `IntList` objects as its input. Since the `head` of the `IntList` type defined here on page 2 is declared **final**, ***your solution must not attempt to assign to any of the*** `head` ***fields.***

```
// Part a.
/** Return the result of applying the swapUp algorithm to L,
 *  non-destructively. */
IntList swapUpList(IntList L) {
    if (L == null) {
        return L;
    } else if (L.head<L.next.head) {

        return new IntList(L.head,
                           swapuplist(l.tail));
    } else {
        return new IntList(L.next.head,
                           swapuplist()

                           );
    }
}
```

```
// Part b. (Do not assign to .head fields)
/** Return the result of applying the swapUp algorithm to L,
 *  destructively.  Does not create new IntList objects. */
IntList dswapUpList(IntList L) {

    if (L==null || L.tail == null) {
        return L;
    }
    IntList holder = new IntList(0, L);
    /* Create no other IntList objects. */
    IntList prev;
    prev = holder;

    while (prev.tail != null) {
        IntList L0 = prev.tail,
                L1 = L0.tail;


        if (L0.tail == L1) {

            L0.tail = L1.tail;

            L1.tail = L0;

            prev = holder;
        } else {

            dswaplist(L0);

        }
    }
    return holder.tail;
}
```

4. [3 points] In the table on the following page, the **Expression** column gives different expressions that could replace the comments in the calls to `System.out.print` in the `Animal` and `Bee` classes. Fill in the **Prints** column with the value these statements would print when filled with the indicated expressions. Sometimes, a given print statement will only be reached for certain replacements of other comments. In those situations, consider only the cases where the print statement actually is reached. When an answer differs depending on what expressions go into other print statements, indicate this with a question mark (?). There are no compilation or runtime errors.

```
class Animal {
    int x = 1;
    static int y = 2;
    int f() {
        System.out.print(/* A */); return 3;
    }
    static int g() {
        System.out.print(/* B */); return 4;
    }
    void h(Animal q, Insect r, Bee s) {
        System.out.print(/* C */);
    }
}

class Insect extends Animal {
    int x = 5;
    static int y = 6;
}

class Bee extends Insect {
    int x = 8;
    static int y = 9;
    int f() {
        System.out.print(/* D */); return 11;
    }
    static int g() { return 10; }
}

class Question {
    public static void main(String... args) {
        Bee b = new Bee();
        b.h(b, b, b);
    }
}
```

Comment	Expression	Prints
/* A */	x	
/* A */	y	
/* A */	this.x	
/* A */	this.y	
/* A */	((Insect) this).x	
/* A */	((Insect) this).y	
/* B */	y	
/* B */	Insect.y	
/* C */	q.f()	
/* C */	r.f()	
/* C */	((Animal) s).f()	
/* C */	((Animal) s).x	
/* D */	super.f()	
/* D */	x	
/* D */	((Animal) this).g()	

5. [2 points] We would like the ability to compare two `Sq` objects (as in Project 0). Consider the implementation below, which defines `Sq` as implementing the `Comparable` interface. ***Each line of code that might contain a bug has a blank line underneath.*** For each line that you think has a bug, fill in the blank line with the corrected code. Here, we use the non-generic definition of `Comparable` from lecture, rather than the newer, improved `Comparable<T>` version.

```
final class Sq implements Comparable {

    /** Compare two Sq objects by the sizes of the group they are in.
     *  That is, s1.compareTo(s2) returns a negative integer if s1's group
     *  is shorter than s2's group.  Numbered squares have a group size of 0. */
    @Override
    public int compareTo(Sq s1) {

    ____________________________________________________

        Sq other = s1;

        ____________________________________________________
        return this.groupSize() - other.groupSize();

        ____________________________________________________
    }

    private int groupSize() {
        if (_sequenceNum != 0) {

        ____________________________________________________
            return 0;
        } else if (_group == -1) {

        ____________________________________________________
            return 1;
        }
        int count = 0;
        for (Sq L = this; L._successor != null; L = L._successor) {

        ____________________________________________________________
            count += 1;
        }
        return count;
    }
```

Continued

Continued from previous page

Code not shown

```
    /** The first in the currently connected sequence of cells ("group")
     *  that includes this one. */
    private Sq _head;

    /** The group number of the group of which this is a member, if
     *  head == this.  Numbered sequences have a group number of 0,
     *  regardless of the value of _group. Unnumbered one-member groups
     *  have a group number of -1.  If _head != this and the square is
     *  unnumbered, then _group is undefined and the square's group
     *  number is maintained in _head._group. */
    private int _group;

    /** True iff assigned a fixed sequence number. */
    private boolean _hasFixedNum;

    /** The current imputed or fixed sequence number,
     *  numbering from 1, or 0 if there currently is none. */
    private int _sequenceNum;
}
```

6. [4 points] Here is a simplified summary of the abstract `java.io.Reader` class, which you saw in Homework 3:

```
public abstract class Reader implements Readable, Closeable {
   public abstract int read(char[] cbuf, int start, int len)
       throws IOException;
   public abstract void close() throws IOException;
   ...
}
```

Create a new `ShiftReader` class that extends this abstract `Reader` class and shifts the characters read from another Reader that is passed to `ShiftReader's` constructor, as for `TrReader` in the homework.

`ShiftReader` adds the functionality that it circularly left-shifts each set of characters read from it by a certain (non-negative) number of positions before adding them to `cbuf`. It is possible to shift by more than the number of characters read; for example, shifting 3 characters left by 5 is the same as shifting them left by 2. So, for example,

```
ShiftReader sr = new ShiftReader(new StringReader("abcdefghi"), 2);
char[] cbuf = {'X', 'X', 'X', 'X', 'X', 'X' };
sr.read(cbuf, 1, 4);
/* cbuf now contains { 'X', 'c', 'd', 'a', 'b', 'X' } */
sr.read(cbuf, 0, 3);
/* cbuf now contains { 'g', 'e', 'f', 'a', 'b', 'X' } */
sr.read(cbuf, 3, 3);
/* cbuf now contains { 'g', 'e', 'f', 'h', 'i', 'X' } */
```

(The last `read` operation above only reads the two remaining characters. Circularly shifting that two-character sequence by two in effect shifts it by one twice, which has no net effect.)

Your solution should only need to use the methods that are in the simplified summary of the `Reader` class above.

```
import java.io.Reader;   import java.io.IOException;
import static java.lang.System.arraycopy; /* See reference material. */
public class ShiftReader extends Reader {

    ---------------------------------------

    ---------------------------------------

    public ShiftReader(Reader src, int shift) {

        ----------------------------------

        ----------------------------------
    }

    public ____________________________________________________ {

        ------------------------------------------------------------
    }

    public ____________________________________________________

        ___________________________________ {
        char[] temp = new char[len];

        int charsRead = ________________________________________________;

        if (________________________________________________) {

            ------------------------------------------------------------

            arraycopy(_____________________________________________

                      ______________________________________________);

            arraycopy(_____________________________________________

                      ______________________________________________);
        }
        return ____________________________________________________;
    }
}
```